Projet_Statistiques

Mouhamadou_Mansour_BA

2024-11-22

MEMBRES DU GROUPE :

- Mouhamadou Mansour BA (IIA)
- Mame Peuya DIA (IIA)
- Fatou FALL (SEM)
- Khoudia Bintou FALL (CAA)

PLAN

  - Enregistrement des couples d'observations (xi,yi) dans un format adapté pour Python
- 3.2 Analyse de la relation entre yi et xi
  - Tracé des points (xi,yi)
  - Observation d'une possible liaison linéaire
- 3.3 Calcul de la droite des moindres carrés
  - Estimation des coefficients de la droite de régression
- 3.4 Calcul des valeurs estimées de yi
  - Calcul des ordonnées des yi estimés pour chaque xi
- 3.5 Tracé de la droite de régression
  - Ajout de la droite sur le graphique de dispersion
- 3.6 Estimation de pour xi=21
  - Calcul de la valeur estimée de Y
- 3.7 Calcul de l'écart
  - Calcul de l'écart entre la valeur observée et la valeur estimée pour xi=21
- 3.8 Vérification du passage par le point moyen t (x^-, y^-)
  - Discussion sur la généralisation du passage par le point moyen pour toute droite de régression

Exercice 4 : Données COVID-19 au Sénégal

- 4.1 Lecture et nettoyage des données
  - Lecture du fichier regions_cas.csv
  - Nettoyage des noms de régions et conversion de la variable date en type datetime
- 4.2 Convertir la variable date en type datetime, et supprimer toutes les lignes ayant des valeurs manquantes
  - Convertir la variable date date en type datetime
  - Supprimer toutes les lignes ayant des valeurs manquantes
- 4.3 Créer une fonction qui retourne un dataframe à 3 colonnes (date, region, maladesparegion). La dernière colonne contiendra le nombre de malades de covid-19 par régions aux différentes dates données
- 4.4 Estimation de
- 4.5 Utilisation d'un test statistique pour vérifier si la variable maladesparegion suit une loi de Poisson
- 4.6 Estimation de r et de p.
- 4.7 Création d'une fonction CarteRegions(madate) qui affiche la carte choroplèthe des régions en utilisant le nombre de malades.

Conclusion

INTRODUCTION :

Ce document présente les solutions aux exercices de statistiques mathématiques pour le Master 1 en Sciences des données à l'Université Iba Der THIAM de Thiès. Ce projet comprend des simulations de lois de probabilité, une estimation par la méthode de Monte Carlo, une analyse de régression linéaire, et une étude de données COVID-19 au Sénégal.

Exercice 1 : Simulation de lois

1.1 Simulation de la loi binomiale B(30, 0.5)

Dans cette première étape, nous simulons un échantillon de taille 10000 suivant une loi binomiale avec 30 essais et une probabilité de succès de 0,5.

- Génération de l'échantillon

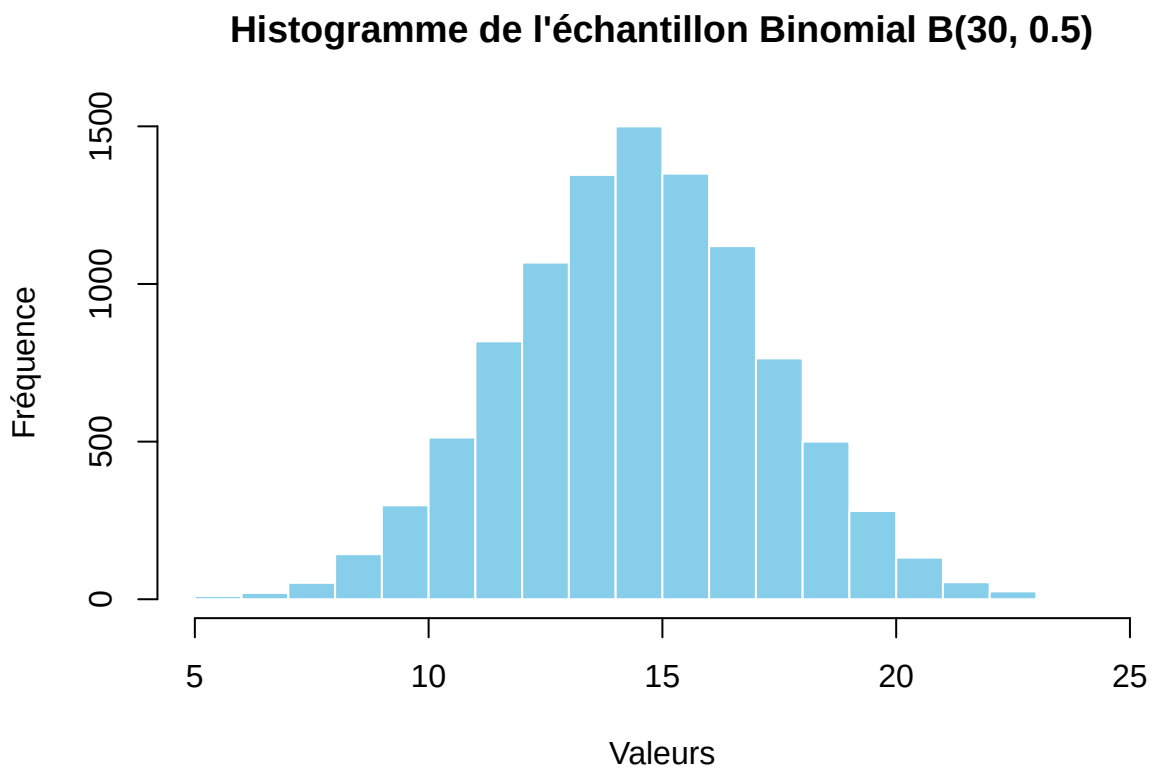
L'objectif est de simuler un échantillon de taille 10 000 provenant d'une distribution binomiale. La loi binomiale B(n,p) décrit le nombre de succès dans n essais indépendants, chacun ayant une probabilité de succès p Dans cet exercice, nous utilisons n=30 (le nombre d'essais) et p=0.5 (la probabilité de succès dans chaque essai).La fonction **`rbinom`** *de R permet de générer cet échantillon.*

```
# Simulation d'un échantillon de taille 10000 suivant une loi binomiale B(30, 0.5)
sample_binomial <- rbinom(10000, size = 30, prob = 0.5)
```

- Tracé de l'histogramme de l'échantillon

Une fois l'échantillon généré, nous traçons un histogramme pour visualiser la distribution des données. L'histogramme est un graphique qui montre la répartition des valeurs observées dans l'échantillon.Chaque barre de l'histogramme représente le nombre d'observations qui tombent dans un intervalle donné.

```
# Tracé de l'histogramme de l'échantillon
hist(sample_binomial, main = "Histogramme de l'échantillon Binomial B(30, 0.5)",
     xlab = "Valeurs", ylab = "Fréquence", col = "skyblue", border = "white")
```



Dans cet histogramme, l'axe des abscisses représente les différentes valeurs possibles du nombre de succès dans les 30 essais (allant de 0 à 30), tandis que l'axe des ordonnées représente la fréquence (ou le nombre d'occurrences) de chaque valeur dans l'échantillon.

- Interprétation de l'histogramme

L'histogramme de la loi binomiale devrait présenter une forme en cloche, symétrique autour de la moyenne (15 dans ce cas), représentant le nombre attendu de succès. Étant donné 30 essais et une probabilité de succès de 0.5, la majorité des échantillons devraient se concentrer autour de cette valeur de 15. Les valeurs extrêmes, proches de 0 ou de 30, seront moins fréquentes, ce qui est caractéristique d'une distribution binomiale.

1.2 Simulation de la loi normale N(3, 0.9)

Nous simulons un échantillon de taille 10 000 suivant une loi normale avec une moyenne de 3 et un écart-type de 0,9, puis traçons la fonction de densité de l'échantillon.

- Génération de l'échantillon

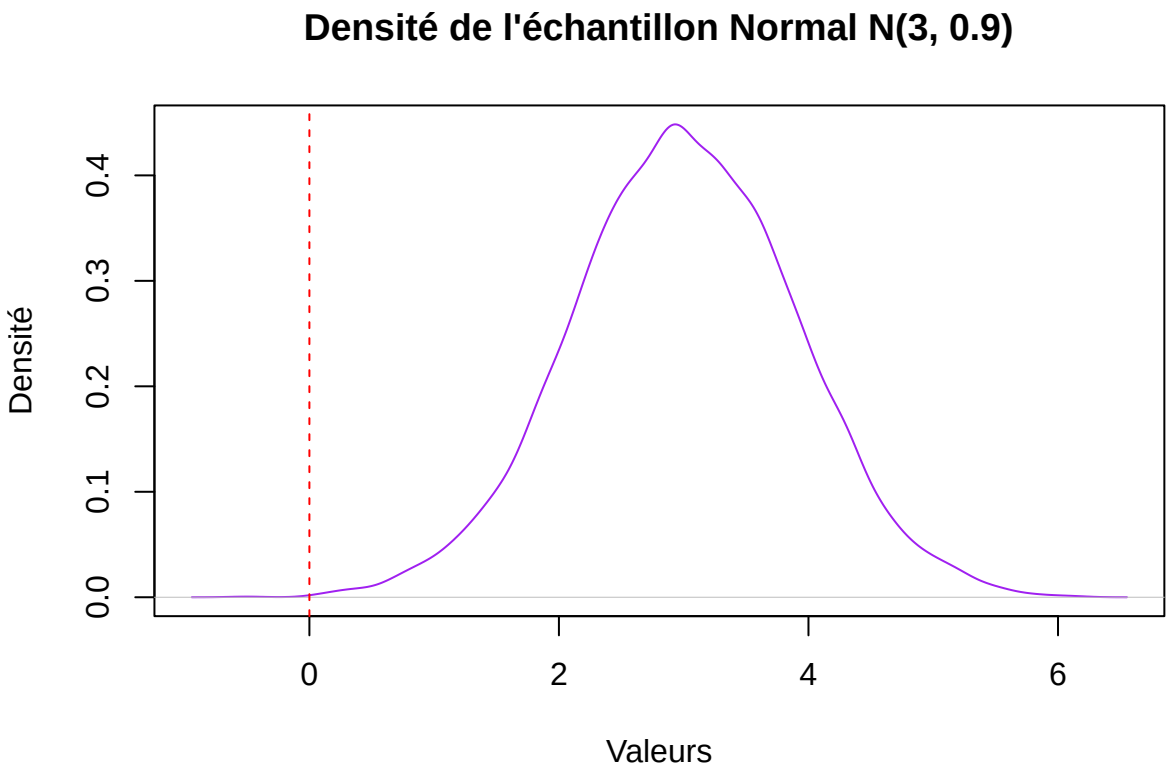
Nous générons un échantillon de taille 10 000 suivant la loi normale N(3, 0.9). La fonction `rnorm` *de R permet de générer cet échantillon.*

```
# Simulation d'un échantillon de taille 10000 suivant une loi normale N(3, 0.9)
sample_normal <- rnorm(10000, mean = 3, sd = 0.9)
```

- Tracé de la fonction de densité de l'échantillon

Une fois l'échantillon généré, nous traçons la fonction de densité qui nous permettra de visualiser la répartition des valeurs dans l'échantillon, nous ajoutons également une ligne pour indiquer l'intervalle contenant 0.

```
# Tracé de la densité de l'échantillon
plot(density(sample_normal), main = "Densité de l'échantillon Normal N(3, 0.9)",
     xlab = "Valeurs", ylab = "Densité", col = "purple")
abline(v = 0, col = "red", lty = 2) # Ajout d'une ligne pour l'intervalle contenant 0
```



- *Interprétation de l'histogramme*

*Le graphique de densité montre la distribution de l'échantillon simulé. La ligne rouge représente un intervalle contenant 0. Comme la loi normale est symétrique, on s'attend à ce que la majorité des valeurs se situent autour de la moyenne 3, avec une dispersion définie par l'écart-type de 0.9.**

1.3 Simulation de la loi du X^2 avec 20 degrés de liberté

Tracer la fonction de densité de l'échantillon obtenu. Choisir un intervalle contenant 0 pour domaine de représentation.

Enfin, nous simulons un chantillon de taille 10 000 suivant une loi du chi-carré avec 20 degrés de liberté et traçons la fonction de densité de l'échantillon.

- *Génération de l'échantillon*

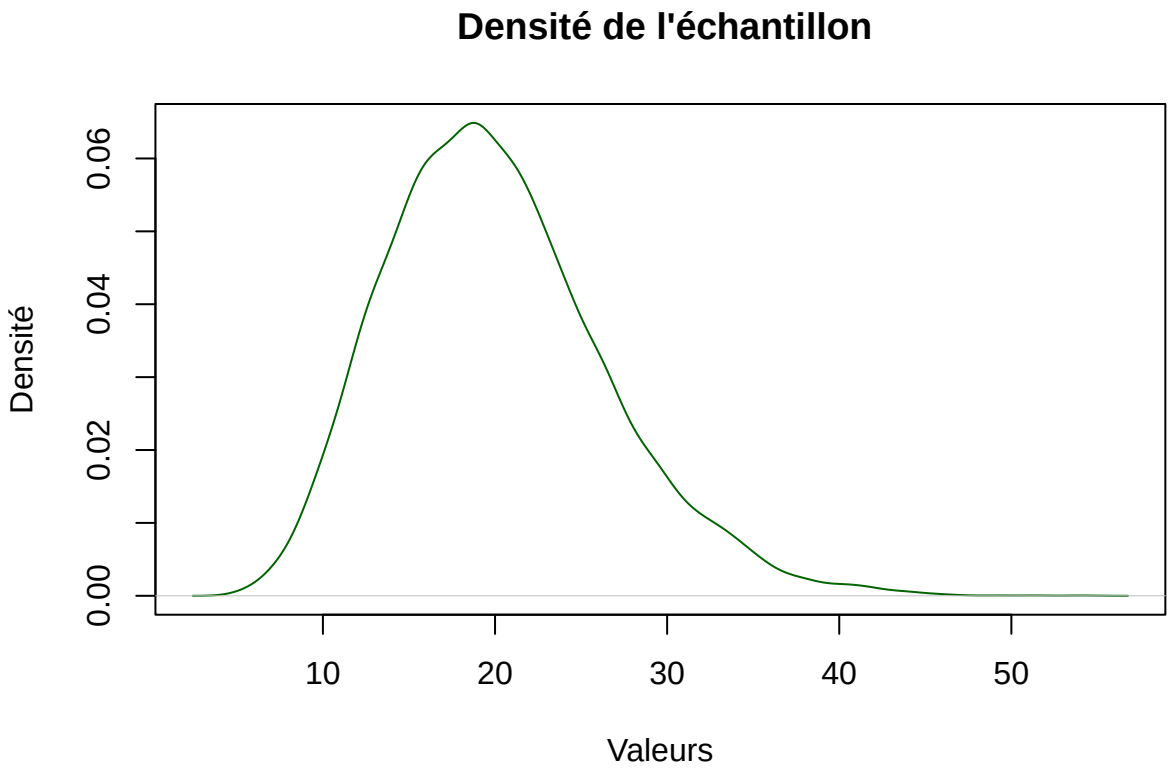
Nous générons un échantillon de taille 10 000 suivant la loi du chi-carré avec 20 degrés de liberté. La fonction rchisq permet de générer cet échantillon.

```
# Simulation d'un échantillon de taille 10000 suivant une loi du ² avec 20 degrés de liberté
sample_chi2 <- rchisq(10000, df = 20)
```

- *Tracé de la fonction de densité de l'échantillon*

Nous traçons ensuite la fonction de densité de l'échantillon pour observer la répartition des valeurs. Comme la loi du chi-carré est asymétrique et prend uniquement des valeurs positives, l'intervalle contenant 0 sera visible sur le graphique.

```r
# Trac\'{e}de la densité de l'\'{e}chantillon
plot(density(sample_chi2), main = " Densité de l'échantillon",
     xlab = "Valeurs", ylab = "Densité", col = "darkgreen")
abline(v = 0, col = "red", lty = 2) # Ligne pour l'intervalle contenant 0
```



- Interprétation de l'histogramme

Le graphique montre que la distribution du chi-carré est fortement asymétrique, avec une forte concentration de valeurs près de 0 et une longue traîne du côté des valeurs plus élevées. La ligne rouge indique l'intervalle contenant 0, ce qui est pertinent car la loi du chi-carré n'a pas de valeurs négatives.

Exercice 2 : Méthode de Monte Carlo

2.1 Estimation de l'intégrale I2

- Utilisation de la méthode de Monte Carlo avec n= 10000

Nous estimons l'intégrale I2 à l'aide de la méthode de `Monte Carlo`*. Cette méthode consiste à générer aléatoirement des points dans l'intervalle [0,1] et à calculer la moyenne des valeurs de la fonction f(x) évaluée en ces points. Avec n = 10 000 simulations, l'estimation obtenue sera comparée à la valeur théorique Pi/4.*

```r
# Nombre de simulations
n <- 10000
```

```r
# Simulation de points uniformément distribués sur [0, 1]
x <- runif(n)

# Fonction à intégrer : sqrt(1 - x^2)
f_x <- sqrt(1 - x^2)

# Estimation de l'intégrale par la méthode de Monte Carlo
I2_estimation <- mean(f_x)

# Affichage de l'estimation
cat("Estimation de I2 par la méthode de Monte Carlo avec n =", n, ":", I2_estimation, "\n")
```

```
## Estimation de I2 par la méthode de Monte Carlo avec n = 10000 : 0.7821071
```

```r
# Valeur théorique de I2
I2_theorique <- pi / 4
cat("Valeur théorique de I2 :", I2_theorique, "\n")
```

```
## Valeur théorique de I2 : 0.7853982
```

2.2 Graphique d'évolution de l'estimation

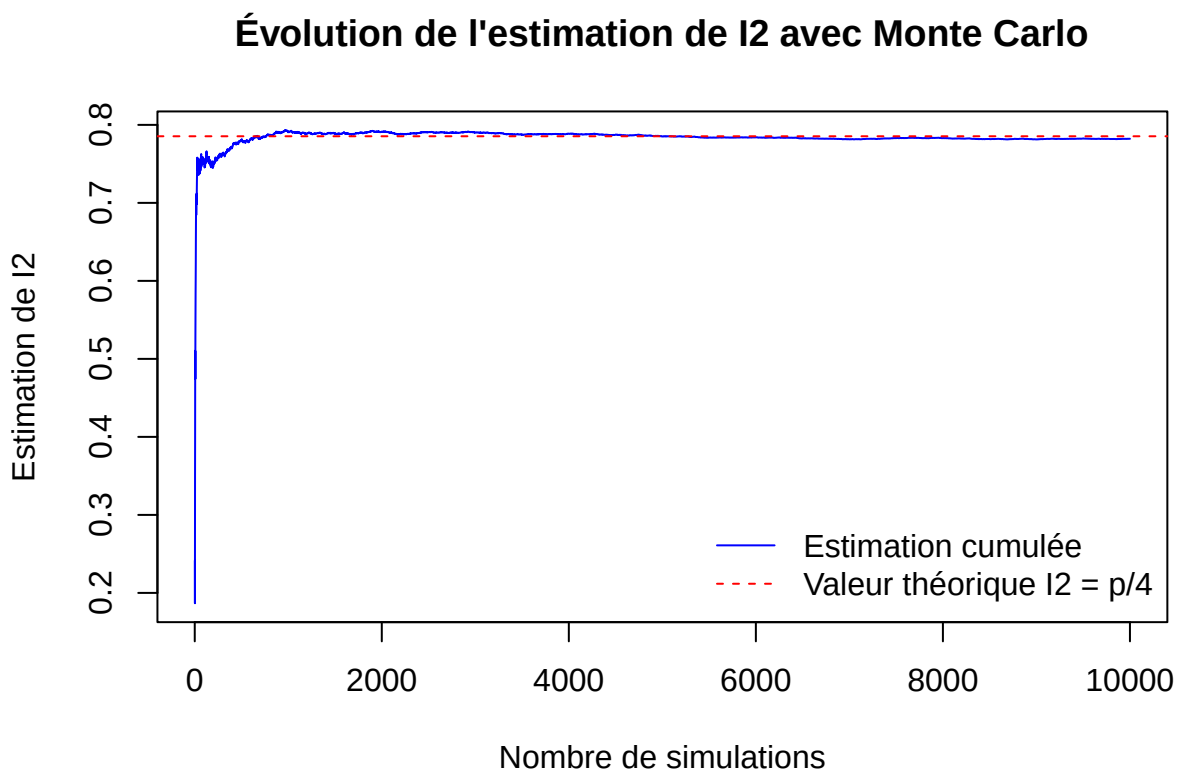
- Tracé de la convergence de l'estimation en fonction de

- Vérification de la cohérence avec la valeur théorique I2 = Pi/4

Pour cette partie, nous visualisons comment l'estimation de I2, obtenue par **`la méthode de Monte Carlo`***, évolue à mesure que le nombre de simulations augmente. L'objectif est de vérifier la convergence de cette estimation vers la valeur théorique I2 = Pi/4.*

Le graphique montre la progression de l'estimation cumulée après chaque simulation. Une ligne horizontale représentant la valeur théorique Pi/4 est ajoutée comme référence visuelle pour comparer les estimations successives. Cette approche permet de vérifier la cohérence de la méthode utilisée et d'observer comment la précision s'améliore avec un plus grand nombre de simulations.

```r
# Visualisation de l'évolution de l'estimation avec n croissant
estimation_progression <- cumsum(f_x) / (1:n)
plot(1:n, estimation_progression, type = "l", col = "blue",
     xlab = "Nombre de simulations", ylab = "Estimation de I2",
     main = "Évolution de l'estimation de I2 avec Monte Carlo")
abline(h = I2_theorique, col = "red", lty = 2) # Valeur théorique en rouge

# Ajout d'une légende
legend("bottomright", legend = c("Estimation cumulée", "Valeur théorique I2 =  /4"),
       col = c("blue", "red"), lty = c(1, 2), bty = "n")
```



- Interprétation du graphe

Le graphique montre l'évolution de l'estimation de I2 en fonction du nombre de simulations n. La courbe bleue correspond à l'estimation cumulative de I2, tandis que la ligne rouge horizontale représente la valeur théorique I2 = Pi/4.

On observe que l'estimation devient de plus en plus stable à mesure que n augmente, convergeant vers la valeur théorique. Cette stabilité indique la validité de la méthode de Monte Carlo pour estimer des intégrales. Pour n = 10 000, les écarts entre les estimations successives et la valeur théorique deviennent négligeables, démontrant que la précision s'améliore avec un plus grand nombre de simulations.

Exercice 3 : Régression Linéaire

3.1 Préparation des données

Enregistrement des couples d'observations (xi,yi) dans un format adapté pour Python.

- Importons d'abord les bibliothèques nécessaires :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

- Enregistrer les données :

```r
# Créer un data.frame pour regrouper les données
donnees <- data.frame(x = c(18, 7, 14, 31, 21, 5, 11, 16, 26, 29),
                      y = c(55, 17, 36, 85, 62, 18, 33, 41, 63, 87))

# Exporter les données au format CSV
write.csv(donnees, "donnees.csv", row.names = FALSE)

cat("Les données ont été enregistrées dans le fichier 'donnees.csv'.\n")
```

```
## Les données ont été enregistrées dans le fichier 'donnees.csv'.
```

- Affichage du fichier donnees.csv

```python
import pandas as pd
donnees = pd.read_csv("donnees.csv")
print(donnees)
```

```
##     x   y
## 0  18  55
## 1   7  17
## 2  14  36
## 3  31  85
## 4  21  62
## 5   5  18
## 6  11  33
## 7  16  41
## 8  26  63
## 9  29  87
```

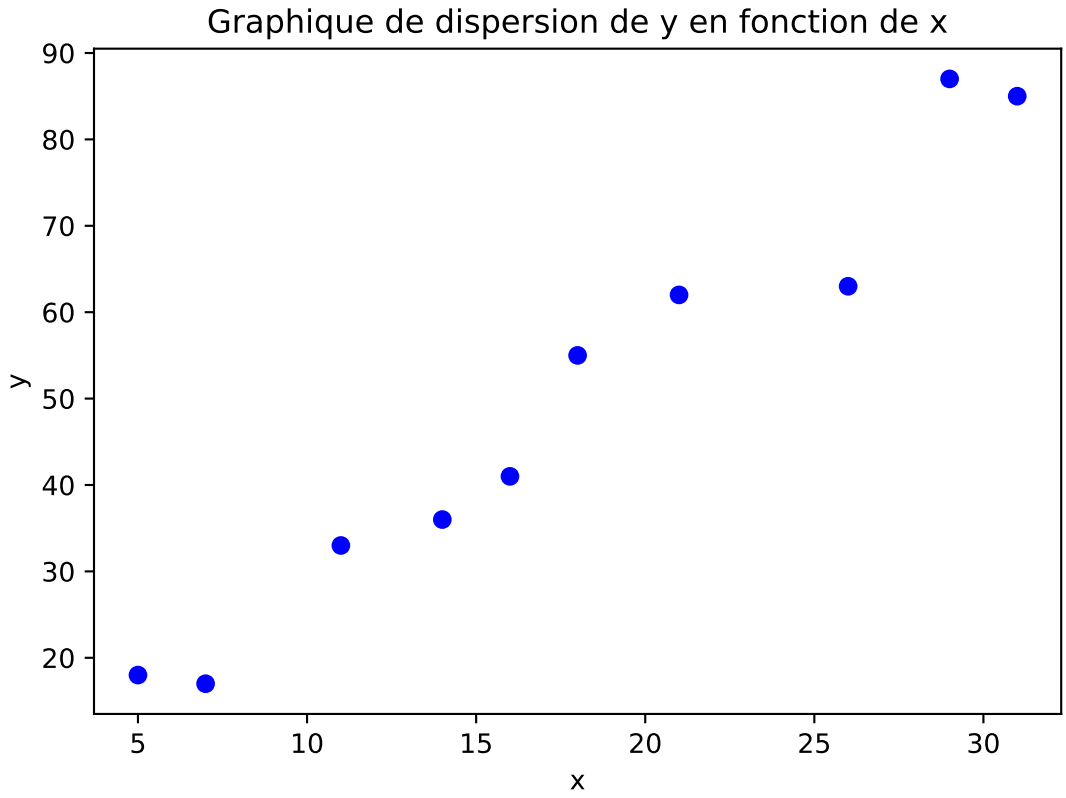
Les vecteurs x et y contiennent les couples d'observations. Cela correspond à la préparation des données pour l'analyse de régression linéaire.

3.2 Analyse de la relation entre yi et xi

- Tracé des points (xi,yi)

- Observation d'une possible liaison linéaire

```python
# Données (x et y)
x = [18, 7, 14, 31, 21, 5, 11, 16, 26, 29]
y = [55, 17, 36, 85, 62, 18, 33, 41, 63, 87]

# Tracer un graphique de dispersion
plt.scatter(x, y, color='blue')

# Ajouter un titre et des labels
plt.title('Graphique de dispersion de y en fonction de x')
plt.xlabel('x')
plt.ylabel('y')

# Afficher le graphique
plt.show()
```



Ce graphique permet de visualiser les points et d'observer une éventuelle relation linéaire entre x et y.

3.3 Calcul de la droite des moindres carrés

- Estimation des coefficients de la droite de régression

```
# Données (x et y)
x = np.array([18, 7, 14, 31, 21, 5, 11, 16, 26, 29]).reshape(-1, 1)
y = np.array([55, 17, 36, 85, 62, 18, 33, 41, 63, 87])

# Ajustement d'une droite de régression linéaire
model = LinearRegression()
model.fit(x, y)
```

```
## LinearRegression()
```

```
# Estimation des coefficients (pente et intercept)
slope = model.coef_[0]
intercept = model.intercept_

# Afficher les coefficients
print("Coefficients de la droite des moindres carrés :")
```

```
## Coefficients de la droite des moindres carrés :
```

```
print(f"Pente (slope) : {slope}")
```

```
## Pente (slope) : 2.7347560975609757
```

```
print(f"Ordonnée à l'origine (intercept) : {intercept}")
```

```
## Ordonnée à l'origine (intercept) : 1.021341463414636
```

Le modèle ajuste la droite des moindres carrés. Les coefficients de régression sont correctement extraits et affichés.

3.4 Calcul des valeurs estimées de yi

- Calcul des ordonnées des yi estimés pour chaque xi

```
# Calculer les valeurs estimées de y
y_estime = model.predict(x)

# Afficher les ordonnées estimées pour chaque x_i
print("Ordonnées estimées pour chaque xi :")
```

```
## Ordonnées estimées pour chaque xi :
```

```
print(y_estime)
```

```
## [50.24695122 20.16463415 39.30792683 85.79878049 58.45121951 14.69512195
##  31.10365854 44.77743902 72.125      80.32926829]
```

Cette étape calcule les yi estimés pour chaque xi. Ces valeurs sont essentielles pour tracer la droite et évaluer la qualité du modèle.

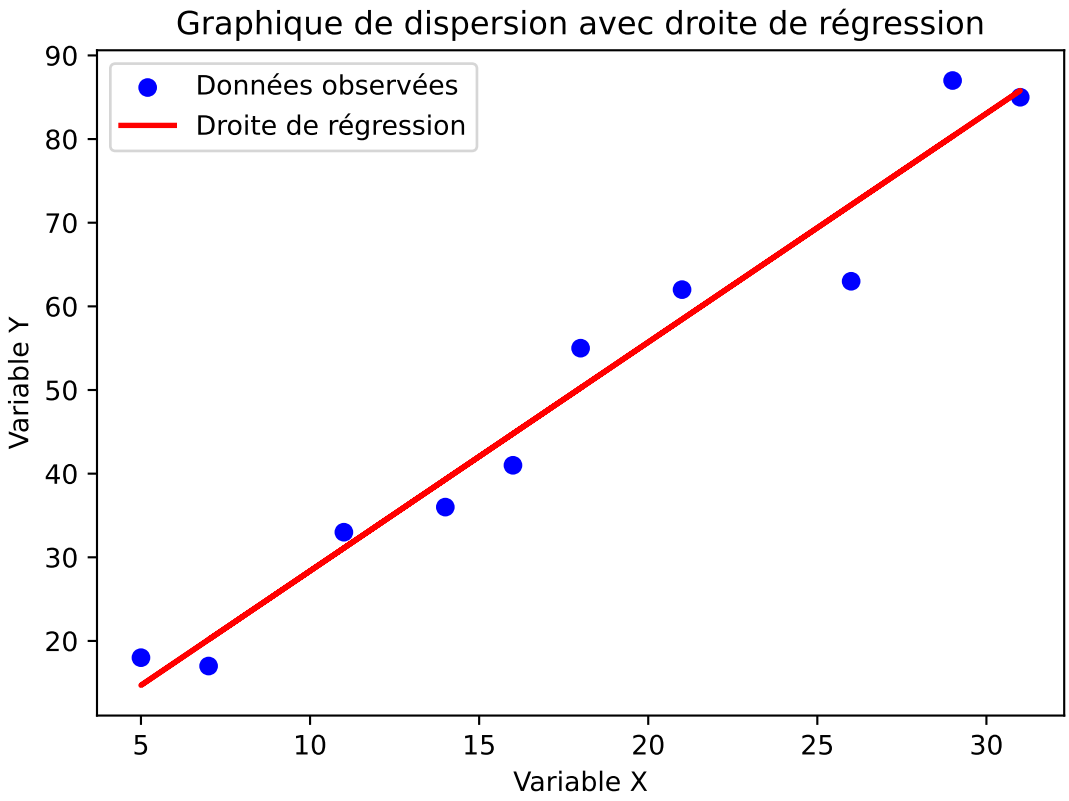
3.5 Tracé de la droite de régression

- Ajout de la droite sur le graphique de dispersion

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Données (x et y)
x = np.array([18, 7, 14, 31, 21, 5, 11, 16, 26, 29]).reshape(-1, 1)
y = np.array([55, 17, 36, 85, 62, 18, 33, 41, 63, 87])

# Ajuster un modèle de régression linéaire
model = LinearRegression()
model.fit(x, y)
```

```
## LinearRegression()
```

```python
# Calculer les valeurs estimées de y (droite de régression)
y_estime = model.predict(x)

# Tracer le graphique de dispersion
plt.scatter(x, y, color='blue', label='Données observées')

# Ajouter la droite de régression
plt.plot(x, y_estime, color='red', linewidth=2, label='Droite de régression')

# Ajouter un titre et des labels
plt.title('Graphique de dispersion avec droite de régression')
plt.xlabel('Variable X')
plt.ylabel('Variable Y')

# Ajouter une légende
plt.legend()

# Afficher le graphique
plt.show()
```



La droite de régression est correctement ajoutée au graphique de dispersion pour visualiser l'ajustement.

3.6 Estimation de Y pour xi=21

- Calcul de la valeur estimée de

```
# Calculer la valeur estimée de y pour xi = 21
x_new = 21
y_estime_21 = slope * x_new + intercept
print(f"Estimation de Y pour xi = 21 : {y_estime_21}")
```

```
## Estimation de Y pour xi = 21 : 58.451219512195124
```

Cette section calcule Y pour xi = 21 à partir du modèle ajusté. C'est une application classique de la prédiction avec un modèle linéaire.

3.7 Calcul de l'écart

- Calcul de l'écart entre la valeur observée et la valeur estimée pour xi = 21

```
# Calculer l'écart entre la valeur observée et la valeur estimée pour x = 21
y_observe_21 = donnees.loc[donnees['x'] == 21, 'y'].values[0]
ecart = y_observe_21 - y_estime_21
print(f"Écart entre la valeur observée et estimée pour x = 21 : {ecart} (L'écart est appelé résidu)")
```

```
## Écart entre la valeur observée et estimée pour x = 21 : 3.5487804878048763 (L'écart est appelé résidu
```

L'écart (ou résidu) est correctement calculé comme la différence entre la valeur observée et la valeur prédite.

3.8 Vérification du passage par le point moyen t (x^-, y^-)

-Discussion sur la généralisation du passage par le point moyen pour toute droite de régression

```
# Calculer les moyennes de x et y
x_bar = donnees['x'].mean()
y_bar = donnees['y'].mean()

# Vérifier si la droite passe par (x̄, ȳ)
y_estime_bar = slope * x_bar + intercept
passe_par_moyen = np.isclose(y_estime_bar, y_bar)
print(f"Point moyen (x̄, ȳ) : ({x_bar}, {y_bar})")
```

```
## Point moyen (x̄, ȳ) : (17.8, 49.7)
```

```
print(f"La droite passe-t-elle par (x̄, ȳ) ? : {passe_par_moyen}")
```

```
## La droite passe-t-elle par (x̄, ȳ) ? : True
```

Cette vérification est importante pour confirmer que la droite de régression passe bien par le barycentre (x^-, y^-), ce qui est une propriété fondamentale des moindres carrés.

Exercice 4 : Données COVID-19 Sénégal

4.1 Lecture et nettoyage des données

-Lecture du fichier regions_cas.csv

```
import pandas as pd

# Charger un fichier Excel
dataframe = pd.read_excel("regions_cas.xlsx")  # Utilise la bonne fonction pour les fichiers Excel

# Afficher un aperçu des données
print(dataframe)
```

```
##           DATE  DAKAR  DIOURBEL  ...  TAMBACOUNDA  THIÈS  ZIGUINCHOR
## 0   2020-03-29     85        26  ...            0     24           3
## 1   2020-03-31    117        26  ...            0     26           3
## 2   2020-04-01    131        26  ...            0     26           3
## 3   2020-04-02    136        26  ...            0     26           3
## 4   2020-04-03    140        26  ...            1     26           3
## ..         ...    ...       ...  ...          ...    ...         ...
## 174 2020-10-14  10465       744  ...          111   1843         563
## 175 2020-10-16  10487       744  ...          111   1854         564
## 176 2020-10-17  10507       744  ...          111   1857         565
## 177 2020-10-19  10533       746  ...          111   1866         566
## 178 2020-10-20  10537       746  ...          111   1886         566
##
## [179 rows x 15 columns]
```

-Nettoyage des noms de régions et mettre la première lettre en majuscule

```
from unidecode import unidecode
import pandas as pd

# Nettoyage des noms des colonnes en enlevant les accents et les points, et en mettant la première lett
dataframe.columns = [unidecode(col).replace('.', '').title() for col in dataframe.columns]

# Affichage des noms des colonnes après nettoyage
print("Les noms des colonnes après nettoyage :")
```

```
## Les noms des colonnes après nettoyage :
```

```
print(dataframe.columns)
```

```
## Index(['Date', 'Dakar', 'Diourbel', 'Fatick', 'Kaffrine', 'Kaolack',
##        'Kedougou', 'Kolda', 'Louga', 'Matam', 'Saint Louis', 'Sedhiou',
##        'Tambacounda', 'Thies', 'Ziguinchor'],
##       dtype='object')
```

```
# Réorganisation des données pour avoir une colonne "Date", une colonne "Region" et une colonne "Malade
données_transformées = pd.melt(dataframe, id_vars=["Date"], var_name="Region", value_name="Malades")

# Affichage des données après la transformation
print("Les données transformées :")
```

```
## Les données transformées :
```

```
print(données_transformées)
```

```
##            Date      Region  Malades
## 0    2020-03-29       Dakar       85
## 1    2020-03-31       Dakar      117
## 2    2020-04-01       Dakar      131
## 3    2020-04-02       Dakar      136
## 4    2020-04-03       Dakar      140
## ...         ...         ...      ...
## 2501 2020-10-14  Ziguinchor      563
## 2502 2020-10-16  Ziguinchor      564
## 2503 2020-10-17  Ziguinchor      565
## 2504 2020-10-19  Ziguinchor      566
## 2505 2020-10-20  Ziguinchor      566
##
## [2506 rows x 3 columns]
```

```
# Sauvegarder les données dans un fichier CSV
données_transformées.to_csv("covid_data.csv", index=False)

# Afficher le contenu du fichier CSV
print(pd.read_csv("covid_data.csv"))
```

```
##            Date      Region  Malades
## 0    2020-03-29       Dakar       85
## 1    2020-03-31       Dakar      117
## 2    2020-04-01       Dakar      131
## 3    2020-04-02       Dakar      136
## 4    2020-04-03       Dakar      140
## ...         ...         ...      ...
## 2501 2020-10-14  Ziguinchor      563
## 2502 2020-10-16  Ziguinchor      564
## 2503 2020-10-17  Ziguinchor      565
## 2504 2020-10-19  Ziguinchor      566
## 2505 2020-10-20  Ziguinchor      566
##
## [2506 rows x 3 columns]
```

```
# Définir la variable 'tab' en tant que copie du DataFrame original
tab = dataframe.copy()

# Afficher 'tab'
print(tab)
```

```
##           Date  Dakar  Diourbel  ...  Tambacounda  Thies  Ziguinchor
## 0   2020-03-29     85        26  ...            0     24           3
## 1   2020-03-31    117        26  ...            0     26           3
## 2   2020-04-01    131        26  ...            0     26           3
## 3   2020-04-02    136        26  ...            0     26           3
## 4   2020-04-03    140        26  ...            1     26           3
## ..         ...    ...       ...  ...          ...    ...         ...
## 174 2020-10-14  10465       744  ...          111   1843         563
## 175 2020-10-16  10487       744  ...          111   1854         564
## 176 2020-10-17  10507       744  ...          111   1857         565
## 177 2020-10-19  10533       746  ...          111   1866         566
## 178 2020-10-20  10537       746  ...          111   1886         566
##
## [179 rows x 15 columns]
```

4.2 Conversion de la variable date en type datetime et suppression de toutes les lignes ayant des valeurs manquantes (s'il en existe)

- Conversion de la variable date en type datetime

```
#Convertir la variable date en datetime
#Utilisons la fonction pd.to_datetime pour convertir la date
données_transformées['Date'] = pd.to_datetime(données_transformées['Date'], errors='coerce')  # Utilisa
print(données_transformées)
```

```
##           Date      Region  Malades
## 0    2020-03-29       Dakar       85
## 1    2020-03-31       Dakar      117
## 2    2020-04-01       Dakar      131
## 3    2020-04-02       Dakar      136
## 4    2020-04-03       Dakar      140
## ...         ...         ...      ...
## 2501 2020-10-14  Ziguinchor      563
## 2502 2020-10-16  Ziguinchor      564
## 2503 2020-10-17  Ziguinchor      565
## 2504 2020-10-19  Ziguinchor      566
## 2505 2020-10-20  Ziguinchor      566
##
## [2506 rows x 3 columns]
```

- Supprimer toutes les lignes ayant des valeurs manquantes (s'il en existe)

```
#Utilisons la méthode `dropna` pour supprimer les lignes qui ont des valeurs manquantes
données_transformées = données_transformées.dropna()
print(données_transformées)
```

```
##           Date      Region  Malades
## 0    2020-03-29       Dakar       85
## 1    2020-03-31       Dakar      117
## 2    2020-04-01       Dakar      131
## 3    2020-04-02       Dakar      136
```

```
## 4    2020-04-03       Dakar      140
## ...         ...         ...      ...
## 2501 2020-10-14  Ziguinchor      563
## 2502 2020-10-16  Ziguinchor      564
## 2503 2020-10-17  Ziguinchor      565
## 2504 2020-10-19  Ziguinchor      566
## 2505 2020-10-20  Ziguinchor      566
##
## [2506 rows x 3 columns]
```

4.3 Création d'une fonction qui retourne un dataframe à 3 colonnes (date, region, maladesparegion). La dernière colonne contiendra le nombre de malades de covid-19 par régions aux différentes dates données.

```
import pandas as pd
import unidecode
#Créons une fonction qui retourne un dataframe à trois colonnes date,region et maladesparregion:

def transformation(données_transformées):

    # Utilisation de melt pour transformer les régions en ligne
    données_transformées = pd.melt(dataframe, id_vars=["Date"], var_name="Region", value_name="maladespa

    return données_transformées


données_nouvelles = transformation(données_transformées)
print(données_nouvelles)
```

```
##            Date      Region  maladesparregion
## 0    2020-03-29       Dakar                85
## 1    2020-03-31       Dakar               117
## 2    2020-04-01       Dakar               131
## 3    2020-04-02       Dakar               136
## 4    2020-04-03       Dakar               140
## ...         ...         ...               ...
## 2501 2020-10-14  Ziguinchor               563
## 2502 2020-10-16  Ziguinchor               564
## 2503 2020-10-17  Ziguinchor               565
## 2504 2020-10-19  Ziguinchor               566
## 2505 2020-10-20  Ziguinchor               566
##
## [2506 rows x 3 columns]
```

4.4 Estimer de

```
#Calculons la moyenne de la colonne maladesparregion:
moy = données_nouvelles['maladesparregion'].mean()
print( "  =",moy)
```

```
##   = 534.425778132482
```

4.5 Utilisation d'un test statistique pour vérifier si la variable maladesparegion suit une loi de Poisson

```
import numpy as np
import scipy.stats as stats

# Estimation du paramètre   à partir de la moyenne des données
moy = données_nouvelles['maladesparregion'].mean()
estimation = moy  #   est estimé par la moyenne des malades

# Définir le nombre de classes (bins) pour l'histogramme
nbre_classe = np.arange(données_nouvelles['maladesparregion'].min(), données_nouvelles['maladesparregion

# Calcul des fréquences observées dans les différentes classes
observed_frequencies, bin_edges = np.histogram(données_nouvelles['maladesparregion'], bins=nbre_classe)

# Calcul des fréquences théoriques attendues pour chaque bin sous l'hypothèse de Poisson
expected_frequencies = []
for i in range(len(nbre_classe) - 1):
    # Calcul de la probabilité d'avoir un nombre de malades dans chaque intervalle
    prob_range = stats.poisson.cdf(nbre_classe[i+1], estimation) - stats.poisson.cdf(nbre_classe[i], est
    expected_frequencies.append(prob_range)

# Normalisation pour avoir la même taille que les fréquences observées
expected_frequencies = np.array(expected_frequencies) * len(données_nouvelles)

# Ajouter une petite constante pour éviter les zéros dans les fréquences attendues
# Cela permet d'éviter la division par zéro ou les erreurs dans le test du chi carré
expected_frequencies = np.maximum(expected_frequencies, 1e-10)

# Effectuer le test du chi carré
chi2_stat, p_value = stats.chisquare(observed_frequencies, expected_frequencies)

# Afficher les résultats
print(f"Statistique du test chi carré : {chi2_stat}")
```

```
## Statistique du test chi carré : 976086011197701.8
```

```
print(f"Valeur p du test : {p_value}")
```

```
## Valeur p du test : 0.0
```

```
# Interpréter le résultat
if p_value < 0.05:
    print("Nous rejetons l'hypothèse nulle : les données ne suivent pas une loi de Poisson.")
else:
    print("Nous ne rejetons pas l'hypothèse nulle : les données suivent probablement une loi de Poisson
```

```
## Nous rejetons l'hypothèse nulle : les données ne suivent pas une loi de Poisson.
```

4.6 Estimation de r et de p

```
import numpy as np

# Calcul de la moyenne et de la variance des données
moy = données_nouvelles['maladesparregion'].mean()
var = données_nouvelles['maladesparregion'].var()

# Estimation de p et r
p = moy / var
r = moy* p / (1 - p)

print(f"Estimation du paramètre p : {p}")
```

```
## Estimation du paramètre p : 0.00018761959852824805
```

```
print(f"Estimation du paramètre r : {r}")
```

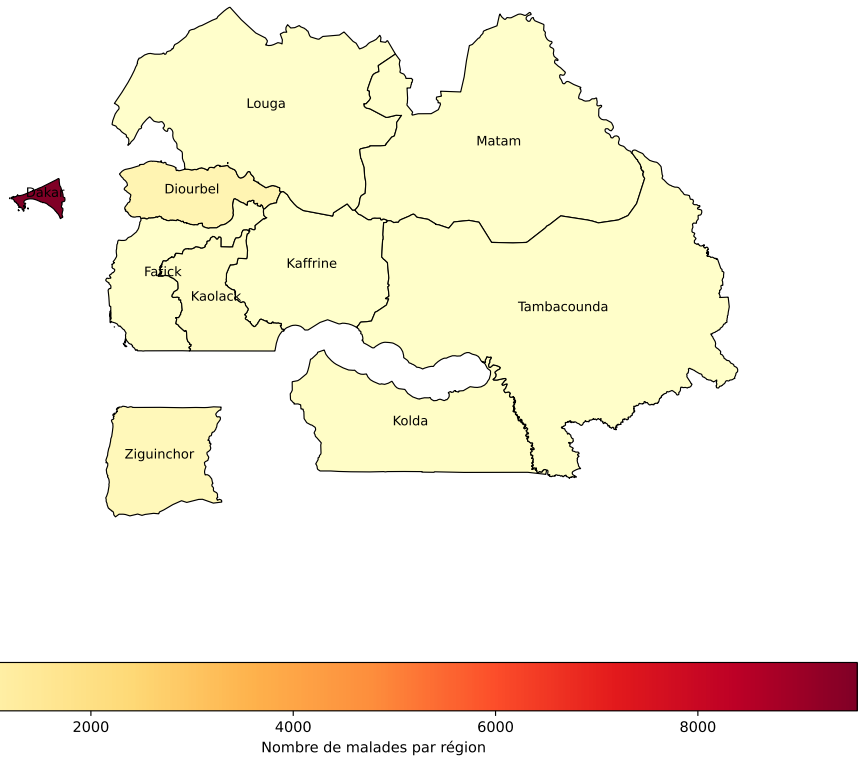
```
## Estimation du paramètre r : 0.10028756584920484
```

4.6 Création de la fonction CarteRegions(madate) qui affiche la carte choroplèthe des régions en utilisant le nombre de malades

Installons les bibliotheques geopandas

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

def CarteRegions(madate, geojson_file="gadm41_SEN_1.json", data_file="covid_data.csv"):

    # Charger les données géographiques
    geo_data = gpd.read_file(geojson_file)

    # Charger les données des malades
    data = pd.read_csv(data_file)
```

```
    # Filtrer les données pour la date donnée
    data_date = data[data["Date"] == madate]

    # Vérifier si des données existent pour la date donnée
    if data_date.empty:
        print(f"Aucune donnée disponible pour la date : {madate}")
        return

    # Normaliser les noms des régions
    geo_data["Region"] = geo_data["NAME_1"].str.strip().str.title()
    data_date.loc[:, "Region"] = data_date["Region"].str.strip().str.title()
    # Fusionner les données géographiques avec les données des malades
    geo_merged = geo_data.merge(data_date, on="Region", how="left")

    # Créer une figure pour la carte
    fig, ax = plt.subplots(1, 1, figsize=(12, 10))

    # Tracer la carte avec les données des malades
    geo_merged.plot(
        column="Malades",
        cmap="YlOrRd",
        linewidth=0.8,
        ax=ax,
        edgecolor="black",
        legend=True,
        legend_kwds={"label": "Nombre de malades par région", "orientation": "horizontal"}
    )

    # Ajouter les noms des régions
    for _, row in geo_merged.iterrows():
        if not pd.isna(row["Malades"]):  # Vérifier qu'il y a des données
            x, y = row["geometry"].centroid.x, row["geometry"].centroid.y
            ax.annotate(
                text=row["Region"],
                xy=(x, y),
                horizontalalignment="center",
                fontsize=9,
                color="black"
            )

    # Ajouter un titre
    ax.set_title(f"Carte des malades de COVID-19 au Sénégal pour la date : {madate}", fontsize=15)

    # Supprimer les axes
    ax.axis("off")

    # Afficher la carte
    plt.show()

# Exemple d'appel de la fonction
CarteRegions("2020-08-31")
```


Carte des malades de COVID-19 au Sénégal pour la date : 2020-08-31
Louga
Matam
Diourbel
Kaffrine
Fatick
Kaolack
Tambacounda
Kolda
Ziguinchor
2000
4000
6000
8000
Nombre de malades par région